const Expr\* factorize(const Expr\* e)

{

variable<const Expr\*> e1, e2, e3, e4;

*Match*(n)

{

*When*(type<Plus>(

type<Times>(e1,e2),

type<Times>(e3 |= e1==e3,e4)

))

return new Times(e1, new Plus(e2,e4));

*When*(type<Plus>(

type<Times>(e1,e2),

type<Times>(e3,e4 |= e2==e4)

))

return new Times(new Plus(e1,e3), e4);

*Otherwise*()

return e;

}

*EndMatch*

}

double power(double x, int n)

{

variable<int> m;

*Match*(n)

{

*When*(0) return 1.0;

*When*(1) return x;

*When*(2\*m) return sqr(power(x,m));

*When*(2\*m+1) return x\*sqr(power(x,m));

}

*EndMatch*

}

template <typename T>

using Cartesian = view<std::complex<T>>;

template <typename T>

using Polar = view<std::complex<T>, polar>;

std::complex<double> c; double a,b,r,f;

*Match*(c)

{

*When*(type< complex<double>>(a,b)) // default

*When*(type<Cartesian<double>>(a,b)) // same as above

*When*(type< Polar<double>>(r,f)) // view

}

*EndMatch*